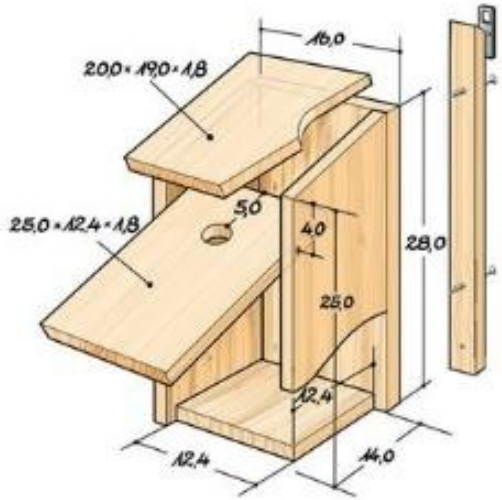
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Programmiervorkurs
Wintersemester 20/21

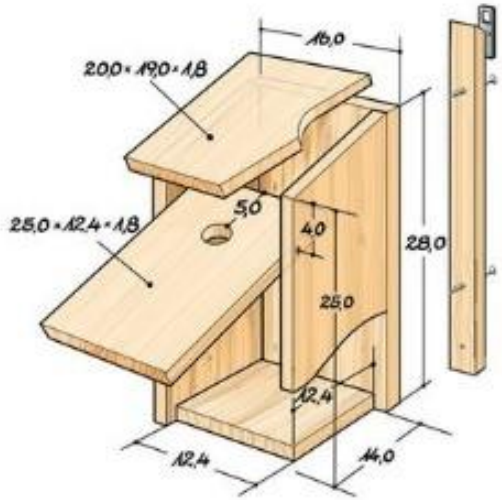
Tag 3



Java

Agenda

- Workshops zur Wiederholung
- Tutorium
  - Objektorientierte Programmierung
- Mäxchen 3.0 & Aufgabe



Modulo Operator

In der Informatik gibt es vielfältige Anwendungsfälle (Verschlüsselung, ...) in denen der Rest einer Division von Interesse ist.

Beispiel normale Division:

9 / 4 = 2,25

Beispiel Division mit Rest:

9 / 4 = 2 Rest 1

Zur Ermittlung des Rests aus einer Division gibt es in Java den Modulo-Operator "%":

```
int r = 9 % 4;    // r wird auf 1 gesetzt
```

XOR-Operator

Analog zu den Operatoren UND und ODER gibt es in Java auch einen XOR-Operator für ein EXKLUSIVES ODER, der Ausdrücke gemäß der Wahrheitstabelle auswertet. Der XOR-Operator wertet zwei Variablen **`a`** und **`b`** mit **`a ^ b`** für aus.

Beispielcode:

```
boolean a = true;
boolean b = false;
if (a ^ b) {
  System.out.println(
    "a oder b ist wahr,
    jedoch nicht beide!");
}
```

Wahrheitstabelle:

a	b	a ^ b
false	false	false
false	true	true
true	false	true
true	true	false

Typecast

Java unterstützt die Umwandlung von Variablen oder Werten in andere Datentypen. Untereinander kompatible Datentypen können so in einen anderen Datentyp umgewandelt werden. Für die Umwandlung wird der Typecast-Operator **`(ZielDatentyp)`** verwendet.

Folgendes Beispiel demonstriert die Umwandlung eines Integers in einen Short:

```
// Legt einen Integer an:
int zahl = 2;
// Weist einem Short den Wert eines Integers zu
short nummer = (short)zahl;
```

Ausführliche Informationen: https://docs.oracle.com/javase/specs/jls/se8/html/jls-5.html

Auswertungsreihenfolge der Operatoren

Werden mehrere Operatoren in einem Ausdruck verwendet, werden diese in folgender Reihenfolge ausgewertet.

Operator	Rang	Typ	Beschreibung
!	1	boolean	logisches Komplement
(Typ)	1	jeder	Typecast
* ,/, %	2	arithmetisch	Multiplikation, Division, Rest
+, -	3	arithmetisch	Addition, Subtraktion
<, <=, >, >=	5	arithmetisch	numerische Vergleiche
==, !=	6	primitiv	Gleich-/Ungleichheit von Werten
&&	10	boolean	logisches konditionales Und
||	11	boolean	logisches konditionales Oder

Vollständige Liste siehe http://openbook.rheinwerk-verlag.de/javainsel/javainsel_02_004.html

Verkürzte If-Abfrage

Java bietet auch eine verkürzte If-Abfrage, die sich besonders dazu eignet, einer Variable Werte in Abhängigkeit einer anderen Variable zuzuweisen oder Text in Abhängigkeit einer Variable auszugeben.

Beispiele für verkürztes If:

```
int temperatur = 5;
boolean unterNull = (temperatur <= 0 ? true : false);
System.out.println("Es ist " + (temperatur <= 0 ? "unter" : "über") + " 0 Grad");
```

Äquivalente Langform:

```
int temperatur = 5;
boolean unterNull;
if (temperatur <= 0) {
  unterNull = true;
  System.out.println("Es ist unter 0 Grad");
} else {
  unterNull = false;
  System.out.println("Es ist über 0 Grad");
}
```

Aufgabe: Visitenkarte

GRUNDLAGE (PABS)

- Erstellt eine Klasse “Visitenkarte“, welche die folgenden Attribute hat: String name, String street, String plz, String city, String phoneNumber, String birthdate, int age
- Erstellt eine weitere Klasse, die die Werte des Benutzers einliest, dann ein Objekt “Visitenkarte“ anlegt und die Werte darin speichert.
- Erzeugt am Ende eine schöne Ausgabe.

ZUSATZ

- Erstellt eine Klasse AutoQuartettKarte, die die Attribute Name, Länge, Hubraum, Leistung und Höchstgeschwindigkeit besitzt. Verwendet keinen Datentyp zweimal.
- Erzeugt 2 Instanzen der Klasse mit beliebigen Werten und vergleicht diese in Bezug auf ein Attribut eurer Wahl.

```
---------------------------------
Name:                Monika Mustermann
Straße:              Musterstraße 1a
Ort:                 12345 Musterstadt
Telefon:             0123 456789
Geb.-Datum:          01.01.1970
Alter:               45
---------------------------------
```

Methoden

- Objekt kann nicht nur Attribute, sondern auch Methoden enthalten
- Methoden geben Objekt die Möglichkeit, Aktionen auszuführen
- Beispiel: Rechteck → Umfang berechnen

```
public class Rechteck {
   int a;
   int b;

   public int berechneUmfang(){
         int umfang = 2*a + 2*b;
         return umfang;
   }
}
```

Methode berechnet den Umfang und gibt diesen als Integer Wert zurück

Methodenverwendung

```
public class Tag3{

  public static void main(String args[]){
    //Rechteck erzeugen
    Rechteck r1 = new Rechteck();

    //Werte zuweisen
    r1.a = 1;
    r1.b = 4;
    umfang  = r1.berechneUmfang();

    System.out.println(„Umfang ist: “ + umfang);
  }
}
```

```
public class Rechteck {
  int a;
  int b;

  public int berechneUmfang(){
       int umfang = 2*a + 2*b;
       return umfang;
  }
}
```

Sichtbarkeit von Attributen eines Objektes

```
public class Tag3{

  public static void main(String args[]){

    //Rechteck erzeugen
    Rechteck r1 = new Rechteck();

    //Werte zuweisen
    r1.a = 1;
    r1.b = 3;
  }
}
```

```
public class Rechteck {
  private int a;
  private int b;

  public int berechneUmfang(){
        int umfang = 2*a + 2*b;
        return umfang;
  }
}
```

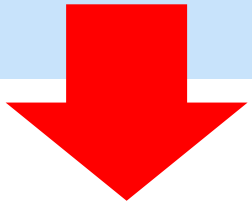
Attribute a und b sind immer noch von außen beeinflussbar, z.B. Negative Werte können zugewiesen werden.

Schutz der Attribute notwendig!

Zugriffsmodifikatoren: public / private

Methoden zum Setzen von Werten

```
public class Tag3{

  public static void main(String args[]){

    //Rechteck erzeugen
    Rechteck r1 = new Rechteck();

    //Werte zuweisen
    r1 .setA(1);
    r1 .setB(4);


  }
}
```



```
public class Rechteck {
  private int a;
  private int b;

  public int berechneUmfang(){
        int umfang = 2*a + 2*b;
         return umfang;
  }

  public void setA(int newA){
    if( newA > 0){
      a = newA;
  }

  public void setB(int newB){
    if( newB > 0){
      b = newB;
  }

}
```

Rechteck soll Kontrolle über Werte haben.
Negative Werte sind nicht erlaubt.

Methoden zum Auslesen von Attributen

```
public class Tag3{

  public static void main(String args[]){
    //Rechteck erzeugen
    Rechteck r1 = new Rechteck();
    //Werte zuweisen
    r1.setA(1);
    r1.setB(4);
    //Werte lesen
    System.out.println(„a: „+r1.getA());
    System.out.println(„a: „+r1.getA());
  }
}
```

Attribute a und b wegen private von außen nicht mehr sichtbar.

Benutzung von Rückgabewerten!

Standard Get-Methode zum Auslesen der Attribute aus einem Objekt

```
public class Rechteck {
  private int a;
  private int b;

  public int berechneUmfang(){
       int umfang = 2*a + 2*b;
         return umfang;
  }

  public void setA(int newA){
    if( newA > 0){
      a = newA;
  }

  public void setB(int newB){
    if( newB > 0){
      b = newB;
  }

  public int getA(){
    return a;
  }

  public int getB(){
    return b;
  }

}
```

Besondere Klassenvariablen

- Variablen, die für **alle** Objekte einer Klasse gelten sollen wird das Stichwort **static** vorangestellt. (z.B. *public static int zahl*).
- Konstanten wird das Stichwort **final** vorangestellt. (z.B. *public static final double PI*) Sie werden nach Konvention groß geschrieben.
- **final** und **static** werden oft zusammen Verwendet

```
public class Warenhaus{

  public static final double MWST = 0.19;

}
```

Aufgabe: Würfelmethoden

GRUNDLAGE

- Erweitert die Klasse Wuerfel um eine Methode spielerWuerfelt(String name), die einmal würfelt und dann folgendes ausgibt: “Spieler <name> hat mit Würfel <farbe> eine <augenzahl> gewürfelt.“
- Zusätzlich braucht die Klasse Würfel noch eine Methode, die sich setFarbe(String farbe) nennt!
- Erweitert die Klasse Würfel um eine Methode getAugenzahlInWorten(), die mittels switch/case-Anweisung die Augenzahlen prüft und die entsprechenden Werte in Worten zurück gibt.

ZUSATZ

- Erweitert die Klasse AutoQuartettKarte um eine Methode erstelleString(boolean b), die alle Attribute eines Autos in einen String zusammenbaut und zurück gibt. Wenn b jedoch falsch ist, soll <XXX> zurückgegeben werden.
- Zusätzlich soll die Klasse eine Methode ausgabe(), die die Methode erstelleString aufruft und deren Ergebnis auf der Konsole ausgibt.
- Füge zwei Konstruktoren hinzu. Einen, der alle Attribute übergeben bekommt, und einen, der keine Attribute übergeben bekommt und daher alles auf leere Strings oder 0 setzt.

Konstruktor

- Manche Objekte haben Eigenschaften, die immer bestehen müssen!
- Diese können direkt beim Erzeugen des Objekts festgelegt werden
  → Konstruktor
- Konstruktor = Methode, die durch new aufgerufen wird (Übergabe von Werten möglich)

```
public class Rechteck {
   private int a;
   private int b;


   public Rechteck(int a, int b){
      this.a = a;
      this.b = b;
   }

   public int berechneUmfang(){
      int umfang = 2*a + 2*b;
      return umfang;
   }
   ...

}
```

```
public class Tag3{

   public static void main(String args[]){
         //Rechteck erzeugen
      Rechteck r1 = new Rechteck(1,4);

   }}
```

Konstruktor

- Zudem können im Konstruktor andere Werte berechnet und gesetzt werden
- Am Beispiel Umfang, dieser kann nach Erstellung mit *a* und *b* berechnet und gesetzt werden.
- Später kann mit *getUmfang()* ohne erneute Berechnung der Umfang benutzt werden

```
public class Tag3{

  public static void main(String args[]){
        //Rechteck erzeugen
    Rechteck r1 = new Rechteck(1,4);

  }}
```

```
public class Rechteck {
  private int a;
  private int b;
  private int umfang;

  public Rechteck(int a, int b){
    this.a = a;
    this.b = b;
    this.umfang = berechneUmfang();
  }

  private int berechneUmfang(){
    int umfang = 2*a + 2*b;
    return umfang;
  }

  public getUmfang(){
    return this.umfang;
  }
  ...

}
```

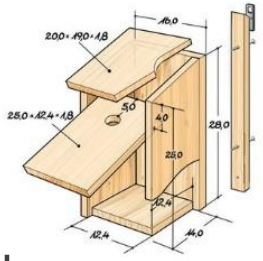
Konstruktor

- **Standardkonstruktor** (z.B. *Haus h = new Haus()*) existiert immer, sofern kein eigener Konstruktor definiert ist.
- **Überladen** (Mehrfachdefinition) möglich, sofern sich diese in Zahl, Datentyp oder Reihenfolge der Übergabeparameter unterscheiden
- Stichwort **this** verweist ausdrücklich auf Klassenattribute (z.B. *this.color*)

```
public class Tag3{

  public static void main(String args[]){
        //Rechteck erzeugen
    Rechteck r1 = new Rechteck(1,4);
    Rechteck r2 = new Rechteck(4);

  }}
```

```
public class Rechteck {
  private int a;
  private int b;
  private int umfang;

  public Rechteck(int a, int b){
    this.a = a;
    this.b = b;
    this.umfang = berechneUmfang();
  }

  public Rechteck(int b){
    this.a = 1;
    this.b = b;
    this.umfang = berechneUmfang();
  }

...

}
```

Klassen Allgemein



Klassenname

Main-Methode

Öffentlich – für jeden von außen sichtbar

Auch ohne Objekt aufrufbar

Keine Rückgabe möglich

Methoden-Name main

Übergabewerte

```
public class ClassName{

    public static void main (String[] args) {

        System.out.println("Hallo Welt!");

    }

}
```

Übungen aus der Aufgabensammlung

1) Erweitert eure Klasse „Würfel“ um einen Konstruktor, mit dem eine Farbe gleich beim erzeugen eines Objektes gesetzt werden muss!

Name	Seite	Schwierigkeit (1-10)	Freiheitsgrad (1-3)	Thema
Drei kleine Schweinchen	20	5	1	Objekte, Methoden, Parameter
Autohaus	25	3	0	Objekte
Mission Impossible	21	5	0	Objekte, Methoden, Parameter, Rückgabewerte
Reaktionsfähigkeit	30	8	3	Recherche, (aber auch Objekte! Alles ist ein Objekt…)

Abschlussaufgabe: Spieler

- Erstellt eine Klasse mit Namen Spieler, von der aus wir später Spieler anlegen, und die die folgenden private Attribute beinhaltet:
  - Name (String name)
  - Alter (int age)
- Zusätzlich sollen für jedes Attribut die Standard Set- und Get-Methoden implementiert werden.
- Auch ein Konstruktor soll vorhanden sein, der beim Erzeugen des Spielers Name und Alter festlegt.
- In einer anderen Klasse:
  - Erstelle 2 Spieler (Gleiche Namen, unterschiedliche Alter).
  - Überprüfe und gebe dann aus, welcher Spieler älter ist.
  - Überprüfe, ob die Spieler gleich heißen und gebe “true“ aus, wenn das der Fall ist.
  - Setze das Alter eines Spielers auf 1 und erhöhe das Alter des anderen um 1. Nutze die Methode von vorher, um auszugeben, wie alt die Spieler in 10 Jahren sind.